MIS Reporting in the Credit Card Industry

Tom Hotard, Acxiom Corporation

ABSTRACT

In credit card acquisition campaigns, it is important to have the ability to keep track of various types of counts. After starting a campaign by pre-defining different populations, MIS reporting systems help capture counts as records progress downstream in the process. Sometimes referred to as a waterfall report, MIS reports come in a variety of flavors. This paper will focus on the creation of MIS edit reports. This type of reporting system is used in a campaign to count the reasons why non-survivors are dropped at gross and net levels from within each population.

The purpose of this paper is to show how SAS® is used to create an MIS edit report for a large credit card issuer, focusing on acquisition campaigns executed in an Oracle® database on a UNIX platform. After first showing an example and illustrating an MIS edit report, the major steps toward generating an MIS edit report will be examined. Highlights of the process will include 1) creating metadata based on a campaign survivability matrix 2) using the metadata to auto-generate IF-THEN and IF-THEN-ELSE statements used to calculate gross and net counts 3) applying the statements to an Oracle table join using the SAS/ACCESS® Pass-Through Facility to Oracle 4) reducing run time by enabling a threaded application using the MP CONNECT tool in SAS/CONNECT®.

INTRODUCTION

Credit card acquisition campaigns are executed for a large issuer of credit cards. Campaign tables are created based on business rules and requirements supplied by the customer. Once tables are created, MIS reports are generated that track the counts of all the reasons why an individual may not be eventually offered a credit card. While SQL is used to create the campaign tables, SAS is used to calculate the MIS reports. In this paper, we first describe in detail how to create the campaign tables from which the MIS report counts are sourced. Next, we describe how SAS code is used to calculate the MIS report counts. Then we build a %MACSUB macro to access Oracle tables, gather counts, and use MP CONNECT to thread the MIS report execution. Finally, we compare run times at various degrees of parallelism.

CREATING THE ORACLE CAMPAIGN TABLES

To better understand the MIS process, it is helpful to first review in general how we develop credit card acquisition campaigns. We start with a database and then create Oracle campaign tables by pulling information from the database, per business requirements, provided by the customer. Basically, a campaign begins by defining target populations and determining whether each individual is included in each population. Next, it is decided for what reasons individuals would be eliminated from particular populations, which leads to identifying the survivors within each population.

Population Flags

To define population flags, we will use Table 1 as an example consisting of ten individuals and three populations. By applying the criteria for each population using SQL, each individual is assigned a dichotomous indicator, where 1 means the individual is included in the particular population and 0 means the individual is not included in the particular population. Each population is treated separately from the others, so it is possible to be included in none or one or more populations.

There are several ways to describe being a member (or not being a member) of a particular population. Belonging to a particular population implies a “hit” and the population flag is “turned on” and assigned a value of 1. Not belonging to a particular population means a “no-hit” and the population flag is “turned off” and assigned a value of 0.

There are two types of population flags that we will use: inclusion and stream. The inclusion population flag is assigned at the very beginning of the process, as shown in Table 1. Later, the next time a population flag is assigned, it will be referred to as a stream, since it is assigned "downstream" from the inclusion. Since streams are assigned downstream, the inclusion could be thought of as being assigned "upstream". In Table 1, the three populations may be generically referred to as POP_A, POP_B, and POP_C, but since we are assigning population flags for the first time, they are referred to as inclusions and each flag is prefixed with an IC. In a next snapshot, taken later downstream, the population flags will be prefixed with an ST, meaning stream, and represented as ST_POP_A, ST_POP_C, and ST_POP_C.

Table 1: Oracle Table Representing Population Inclusion Flags

Individual	IC_POP_A	IC_POP_B	IC_POP_C
1	0	0	1
2	0	1	1
3	1	1	1
4	0	1	0
5	0	0	1
6	1	1	0
7	0	0	1
8	0	1	0
9	0	0	1
10	1	0	1

In Table 1, for example, individual # 10 hits IC_POP_A and IC_POP_C but does not hit IC_POP_B. It should be noted that because individual # 10 is included in two populations, his ultimate survival in one population is independent of his ultimate survival in another population. The record for individual # 10 will not be dropped, even if the flags for POP_A or POP_C or both are later be turned off downstream, after applying additional criteria. Once a population flag is turned off, it can not be turned back on later.

Edit Tags

In a separate step, individuals are tagged for edits based on criteria defined by the customer. In this example, SQL is used to determine whether each individual can be tagged for each of four separate edits (EDIT_1, EDIT_2, EDIT_3, EDIT_4). If an individual is tagged for a certain edit, a 1 is assigned. If not, the edit tag receives a 0. An individual being tagged may lead to an eventual elimination from particular populations.

Table 2: Oracle Table Representing Edit Tags

INDIVIDUAL	EDIT_1	EDIT_2	EDIT_3	EDIT_4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	1	0	1	0
5	0	0	1	1
6	1	1	1	1
7	0	1	0	0
8	1	0	1	0
9	0	0	0	1
10	1	1	1	0

From Table 2, suppose EDIT_1 indicates an invalid zip code. For each individual, if there is an indication of an invalid zip code, then EDIT_1 is turned on (EDIT_1 = 1), otherwise EDIT_1 is turned off (EDIT_1 = 0). For example, individual # 4 is tagged for EDIT_1 and EDIT_3, but not for EDIT_2 and EDIT_4.

The next step is to join the Oracle tables together, as shown in Table 3, displaying all population flags and edits tags, in a single table, for each individual.

Table 3: Oracle Table after Combining Population Flags and Edit Tags

INDIVIDUAL	IC_POP_A	IC_POP_B	IC_POP_C	EDIT_1	EDIT_2	EDIT_3	EDIT_4
1	0	0	1	0	1	1	0
2	0	1	1	0	0	1	1
3	1	1	1	0	1	0	1
4	0	1	0	1	0	1	0
5	1	1	1	0	0	1	1
6	1	1	0	0	0	0	0
7	1	0	1	0	1	0	0
8	0	1	1	1	0	1	0
9	0	0	1	1	1	0	1
10	1	0	1	1	1	1	0

Table 3 only shows the campaign snapshot after the first stage. The next stage will involve determining how population flags will change, if at all, when business decisions concerning the edits tags are applied to the populations. Table 4 shows the unknown population streams (ST_POP_A, ST_POP_B, ST_POP_C) concatenated on the end.

Table 4: Combined Oracle Table with Unknown Population Stream Flags

	POPULATION INCLUSION FLAGS			EDIT TAGS				POPULATION STREAM FLAGS		
INDIVIDUAL	IC_POP_A	IC_POP_B	IC_POP_C	EDIT_1	EDIT_2	EDIT_3	EDIT_4	ST_POP_A	ST_POP_B	ST_POP_C
1	0	0	1	0	1	1	0	0	0	?
2	0	1	1	0	0	1	1	0	?	?
3	1	1	1	0	1	0	1	?	?	?
4	0	1	0	1	0	1	0	0	?	0
5	1	1	1	0	0	1	1	?	?	?
6	1	1	0	0	0	0	0	?	?	0
7	1	0	1	0	1	0	0	?	0	?
8	0	1	1	1	0	1	0	0	?	?
9	0	0	1	1	1	0	1	0	0	?
10	1	0	1	1	1	1	0	?	0	?

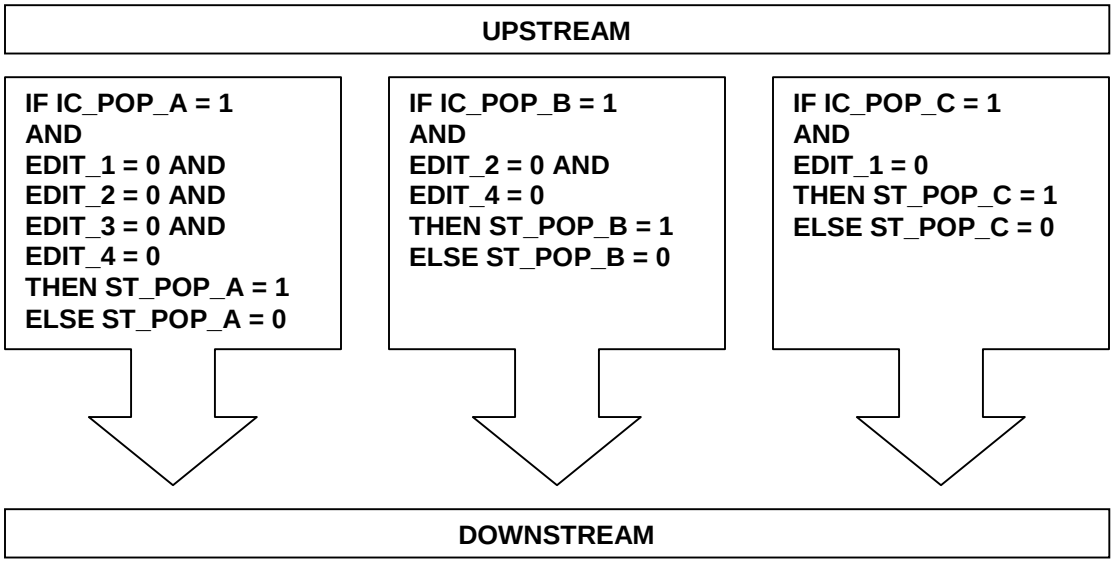
At this point in the campaign, we do not know how the streams will be assigned, except that an individual having a particular inclusion turned off will also have the particular stream also turned off, as shown in Table 4. For individual # 1, we know that since there was a no-hit for POP_A (IC_POP_A = 0) and POP_B (IC_POP_B = 0) that their respective streams flags will be turned off (ST_POP_A = 0 and ST_POP_B = 0). A survivability matrix, shown in Table 5, contains information to determine by what rules stream flags will be turned off within populations.

Table 5: Survivability Matrix

Edit/Pop	POP_A	POP_B	POP_C
EDIT_1	0	-	0
EDIT_2	0	0	-
EDIT_3	0	-	-
EDIT_4	0	0	-

By reading down a particular column in Table 5, we can determine the criteria needed for a population flag, originally turned on as an inclusion, to remain turned on as a stream. For example, by reading down the second column, a particular individual having IC_POP_B = 1 will have ST_POP_B = 1 only if EDIT_2 = 0 and EDIT_4 = 0. The cells for EDIT_1 and EDIT_3 contain a dash and, therefore, are excluded from the stream criteria. Figure 1 shows, for each population, how the survivability matrix is translated into SAS code. Like a sandbox sifter, the survivability matrix is used to filter the populations for each individual.

Figure 1: Applying Stream Criteria Derived from Survivability Matrix



Using Figure 1, values are assigned to the population streams for each individual, as shown in Table 6.

Table 6: Combined Oracle Table with Known Population Stream Flags

	POPULATION INCLUSION FLAGS			EDIT TAGS				POPULATION STREAM FLAGS		
INDIVIDUAL	IC_POP_A	IC_POP_B	IC_POP_C	EDIT_1	EDIT_2	EDIT_3	EDIT_4	ST_POP_A	ST_POP_B	ST_POP_C
1	0	0	1	0	1	1	0	0	0	1
2	0	1	1	0	0	1	1	0	0	1
3	1	1	1	0	1	0	1	0	0	1
4	0	1	0	1	0	1	0	0	1	0
5	1	1	1	0	0	1	1	0	0	1
6	1	1	0	0	0	0	0	1	1	0
7	1	0	1	0	1	0	0	0	0	1
8	0	1	1	1	0	1	0	0	1	0
9	0	0	1	1	1	0	1	0	0	0
10	1	0	1	1	1	1	0	0	0	0

For example, for individual # 7, having only one edit flag turned on is all it takes to turn off the stream for POP_A. And since EDIT_2 has no effect on POP_C, its stream is turned on (ST_POP_C = 1).

MIS REPORT CALCULATION

Now that the all the inclusion and stream flags have been calculated, as shown in Table 6, we can begin focusing on how to calculate the MIS report. We will actually calculate two MIS reports, one showing gross edit counts by population and the other showing net edit counts by population. Each report is typically displayed as an Excel spreadsheet. It should be noted that the MIS process does not turn on or off any population flags. Its job is to count how many times a population flag has been turned off for each edit that may have turned it off. From Table 6, for individual # 3, since the inclusion for POP_A is turned on and later the stream turned off, we know that a count will be recorded on the MIS. The survivability matrix will help to determine the rules by which the MIS reports will be counted.

CALCULATING THE MIS REPORT MANUALLY

We will first calculate the MIS Report manually, based on the results from Table 6 and the survivability matrix in Table 5.

Gross Edit MIS Counts

Per the survivability matrix, the gross edit MIS counts the number of times each edit is turned on when the stream has been turned off. The following bullets are examples of how to populate Table 7 with gross MIS counts calculated from Table 6, while using Table 5 as a guide:

- While only looking at individuals whose IC_POP_A = 1, when reading down the ST_POP_A and EDIT_1 columns together, ST_POP_A = 0 and EDIT_1 = 1 a total of 1 time.
- While only looking at individuals whose IC_POP_A = 1, when reading down the ST_POP_A and EDIT_2 columns together, ST_POP_A = 0 and EDIT_2 = 1 a total of 3 times.
- While only looking at individuals whose IC_POP_A = 1, when reading down the ST_POP_B and EDIT_4 columns together, ST_POP_B = 0 and EDIT_4 = 1 a total of 3 times.

Table 7: MIS Gross Edit Counts by Population Report

Edit/Pop	POP_A	POP_B	POP_C
EDIT_1	1	-	3
EDIT_2	3	1	-
EDIT_3	2	-	-
EDIT_4	2	3	-

Remember, for example, that EDIT_1 has no impact on POP_B, as indicated by a dash in the survivability matrix, and, therefore, that cell in the MIS report will still indicate a dash.

Net Edit MIS Counts

When calculating Net Edit counts, we must respect the hierarchy of the edits. For example, in Table 5, for POP_B, when reading from top to bottom, EDIT_2 is the “best” edit and EDIT_4 is the “second best” edit. Per the survivability matrix, the net edit MIS counts the number of times each edit is the “first” one to have been turned on when the stream has been turned off. The following are examples of how to populate Table 8 with net edit MIS counts calculated from Table 6 while using Table 5 as a guide:

- While only looking at individuals whose IC_POP_A = 1, when reading down the ST_POP_A and EDIT_2 columns together, ST_POP_A = 0 and EDIT_2 = 1 a total of 2 times when a better edit

(EDIT_1) was not turned on at the same time. In other words, when the POP_A stream is turned off after having the inclusion turned on, EDIT_2 = 1 when EDIT_1 = 0 and this happens for 2 individuals.

- While only looking at individuals whose IC_POP_B = 1, when reading down the ST_POP_B and EDIT_2 columns together, ST_POP_B = 0 and EDIT_2 = 1 a total of 1 time. Note that EDIT_1 is not a better edit.
- While only looking at individuals whose IC_POP_B = 1, when reading down the ST_POP_B and EDIT_4 columns together, ST_POP_B = 0 and EDIT_4 = 1 a total of 2 times when the better edit (EDIT_2) is not turned on.

Table 8: MIS Net Edit Counts by Population Report

Edit/Pop	POP_A	POP_B	POP_C
EDIT_1	1	-	3
EDIT_2	2	1	-
EDIT_3	1	-	-
EDIT_4	0	2	-

When comparing Table 7 to Table 8, the gross and net edit MIS counts are always the same for the best edit for a particular population.

CALCULATING THE MIS REPORT WITH SAS

There are three major step involved in creating the MIS report using SAS. After first converting the survivability matrix into a SAS dataset, it will drive the generation of SAS code and other parameters that are turned into macro variables. The final step is building a %MACSUB macro, which will garner access to Oracle using the SAS/ACCESS Pass-Through Facility, execute the MIS count code in a single I/O step, and thread the MIS run for efficiency.

STEP 1: Turn the Survivability Matrix into a SAS Dataset

We first turn the survivability matrix (Table 5) into SAS dataset having 3 variables. This metadata, shown in Table 9, will primarily be used to generate SAS code used to calculate MIS counts and other parameters which are converted into macro variables.

Table 9: MIS Metadata Used by SAS

STREAM	EDIT	VALUE
POP_A	EDIT_1	0
POP_A	EDIT_2	0
POP_A	EDIT_3	0
POP_A	EDIT_4	0
POP_B	EDIT_1	-
POP_B	EDIT_2	0
POP_B	EDIT_3	-
POP_B	EDIT_4	0
POP_C	EDIT_1	0
POP_C	EDIT_2	-
POP_C	EDIT_3	-
POP_C	EDIT_4	-

STEP 2: Generate MIS Count Code and Create Macro Variables

The above metadata is used by the code in Appendix A to generate the IF-THEN and IF-THEN-ELSE statements used to calculate the gross and net counts, respectively.

IF-THEN Statements for MIS Gross Edit Calculation

```
IF IC_POP_A = 1 AND ST_POP_A = 0 THEN DO;
  IF EDIT_1 = 1 THEN GRO_POP_A__EDIT_1 + 1;
  IF EDIT_2 = 1 THEN GRO_POP_A__EDIT_2 + 1;
  IF EDIT_3 = 1 THEN GRO_POP_A__EDIT_3 + 1;
  IF EDIT_4 = 1 THEN GRO_POP_A__EDIT_4 + 1;
END;

IF IC_POP_B = 1 AND ST_POP_B = 0 THEN DO;
  IF EDIT_2 = 1 THEN GRO_POP_B__EDIT_2 + 1;
  IF EDIT_4 = 1 THEN GRO_POP_B__EDIT_4 + 1;
END;

IF IC_POP_C = 1 AND ST_POP_C = 0 THEN DO;
  IF EDIT_1 = 1 THEN GRO_POP_C__EDIT_1 + 1;
END;
```

For example, GRO_POP_A__EDIT_3 stores the count of the number of times EDIT_3 = 1 when IC_POP_A = 1 and ST_POP_A = 0.

IF-THEN-ELSE Statements for MIS Net Edit Calculation

```
IF IC_POP_A = 1 AND ST_POP_A = 0 THEN DO;
        IF EDIT_1 = 1 THEN NET_POP_A__EDIT_1 + 1;
  ELSE IF EDIT_2 = 1 THEN NET_POP_A__EDIT_2 + 1;
  ELSE IF EDIT_3 = 1 THEN NET_POP_A__EDIT_3 + 1;
  ELSE IF EDIT_4 = 1 THEN NET_POP_A__EDIT_4 + 1;
END;

IF IC_POP_B = 1 AND ST_POP_B = 0 THEN DO;
        IF EDIT_2 = 1 THEN NET_POP_B__EDIT_2 + 1;
  ELSE IF EDIT_4 = 1 THEN NET_POP_B__EDIT_4 + 1;
END;

IF IC_POP_C = 1 AND ST_POP_C = 0 THEN DO;
  IF EDIT_1 = 1 THEN NET_POP_C__EDIT_1 + 1;
END;
```

For example, NET_POP_B__EDIT_4 stores the count of the number of times EDIT_4 = 1 when IC_POP_B = 1 and ST_POP_B = 0 and EDIT_4 = 0.

Macro Variables Derived from Metadata

Table 10 lists all the macro variables that are derived from the metadata, and provides macro variable resolutions based on our example. Some macro variables refer to fixed parameters, while others are population-specific and only used for the ith population. Appendix A shows how the Data step uses the CALL SYMPUT routine to assign values to macro variables.

Table 10: Important Macro Variables

Macro Variable	Description	Example of a Particular Macro Variable Resolution when Applied to our Example
&num_streams	Number of populations	3
&a_ic_name	List of all population inclusion flags	IC_POP_A IC_POP_B IC_POP_C
&a_st_name	List of all population stream flags	ST_POP_A ST_POP_B ST_POP_C
&gro_elements	Variables that will store gross MIS counts	GRO_POP_A__EDIT_1 GRO_POP_A__EDIT_2 GRO_POP_A__EDIT_3 GRO_POP_A__EDIT_4 GRO_POP_B__EDIT_2 GRO_POP_B__EDIT_4 GRO_POP_C__EDIT_1

Macro Variable	Description	Example of a Particular Macro Variable Resolution when Applied to our Example
&net_elements	SAS variables used to store net MIS counts	NET_POP_A__EDIT_1 NET_POP_A__EDIT_2 NET_POP_A__EDIT_3 NET_POP_A__EDIT_4 NET_POP_B__EDIT_2 NET_POP_B__EDIT_4; NET_POP_C__EDIT_1
&&groedit&i	IF-THEN statement that calculates gross MIS counts for all edits capable of turning off a particular population stream per the survivability matrix	IF IC_POP_B = 1 AND ST_POP_B = 0 THEN DO; IF EDIT_2 = 1 THEN GRO_POP_B__EDIT_2 + 1; IF EDIT_4 = 1 THEN GRO_POP_B__EDIT_4 + 1; END;
&&netedit&i	IF-THEN-ELSE statement that calculates net MIS counts for all edits capable of turning off a particular population stream per the survivability matrix	IF IC_POP_B = 1 AND ST_POP_B = 0 THEN DO; IF EDIT_2 = 1 THEN NET_POP_B__EDIT_2 + 1; ELSE IF EDIT_4 = 1 THEN NET_POP_B__EDIT_4 + 1; END;

STEP 3: Build %MACSUB Macro

The %MACSUB macro, shown in Appendix B, was constructed in order to package together the code necessary to gain access to Oracle in a multi-processor environment and produce the MIS reports as fast and efficiently as possible.

The major parts to the %MACSUB macro are:

1. MP CONNECT and exploitation of the %SYSLPUT macro statement
2. SAS/ACCESS Pass-Through Facility
3. MIS report counting code

MP CONNECT and Exploitation of the %SYSLPUT Macro Statement

We will take advantage of our multi-processor environment by implementing MP CONNECT and using %SYSLPUT to pass macro variables to other SAS sessions invoked on remote hosts. In this case, the remote hosts are other SAS sessions on the same machine as the local SAS session. In Appendix B, Lines 3 - 33 and 76 will kickoff remote SAS sessions and use %SYSLPUT to send to each the macro variables listed in Table 10. Table 11 shows which macro variables are being used in the remote SAS sessions, prefixed with an "r", along with the macro variable originally created in the local SAS session.

Table 11: Important Macro Variables

Original Macro Variable	Macro Variable Name in Remote SAS Session
&num_streams	&rnum_streams
&a_ic_name	&ra_ic_name
&a_st_name	&ra_st_name
&gro_elements	&rgro_elements
&net_elements	&rnet_elements
&&groedit&i	&&rgroedit&i
&&netedit&i	&&rnetedit&i
&mmod	&rmmod
&flist	&rflist

Additional macro variables in Table 11 but not in Table10, and therefore not created by the code in Appendix A, are the &flist and &mmod macro variables. The &flist is a list of all inclusion, streams and edits fields sourced from the Oracle campaign tables, which was generated using the code shown in Appendix C. The &mmod macro variable is described in the next sub-section.

SAS/ACCESS Pass-Through Facility

We connect to the Oracle campaign tables using the SAS/ACCESS Pass-Through Facility (lines 39 - 50 in Appendix B). The PROC SQL procedure includes the tables to be joined, the Oracle source fields stored in the &rflist macro variable, and the application of the mod function. The mod function is used to thread the MIS run by partitioning the Oracle data into separate, non-overlapping portions, each processed by its own remote SAS session. The argument of the mod function determines the number of remote SAS sessions MP CONNECT will spawn. The &mmod macro variable, whose outcome determines the non-overlapping portions of the Oracle source tables, is passed to the remote SAS sessions as &rmmod.

MIS report counting code

In Appendix B, lines 52 - 67 produces the gross and net edit MIS counts by population. This processing is done in a single I/O pass of the data. All the IF-THEN and IF-THEN-ELSE statements are applied to a Data step that reads in the SQL view to the joined Oracle campaign tables. The two arrays on lines 54 and 55 indicate all the inclusion and stream fields, respectively, with array dimension being the number of populations in the campaign. Lines 56 and 57 initialize all count variables that will retain gross and net edit counts, respectively. Lines 58 - 63 is where the counting actually occurs, but is only required when populations, originally turned on as inclusions, are turned off as streams. On line 64 - 66, after the last record is processed, the variables holding all gross and net edit counts are written to a single-record SAS dataset.

MP CONNECT RUN TIME COMPARISONS

Originally, for efficiency purposes, the MIS report SAS program was designed to run in a single I/O step. However, it was also designed to run on a single processor in a multi-processor environment. After implementing MP CONNECT and using the mod function, comparisons were made to see how much faster multi-threaded programs ran compared to the original single-threaded program. We compared run times to be what would have been expected assuming linear scalability. When compared to a single-processor run, linear scalability means expected run time decreases in proportion to the number of processors added to the run. For example, when executing on three processors, the MIS reports should run in one-third the time when compared to the original, single-processor run. Table 12 displays the specifications of the server used to test scalability.

Table 12: Server Specifications

Type of Machine: Compaq GS140
Operating System: Tru64 Unix v4.0.f
Number of Processors: 8
Cycles per Processor: 699 MHz
Amount of RAM: 12 GB
SAS Version: 8.1
Oracle Version: 8i (v8.1.7)
Records in Oracle Driver Table: 90 Million

Since the server has eight processors, eight separate MIS report runs were executed with actual and expected run times recorded in Table 13. Programmatically, as the number of processors increased, so did the mod function argument, which became equal to the number of processors used for a particular run. The original run time of 13 hours and 46 minutes was used as the basis for calculating expected run times assuming linear scalability.

Table 13: Actual and Expected Change in Run Time as Number of Processors Increases

Number of processors	Actual Run Time	Actual Scalability Compared to Single-Processor Run Time	Expected Scalability Compared to Single-Processor Run Time	Expected Run Time Assuming Linear Scalability
1	13:46	1:1	1:1	13.46
2	6:40	2.1:1	2:1	6:53
3	4:48	2.9:1	3:1	4:35
4	4:05	2.7:1	4:1	3:27
5	3:49	3.6:1	5:1	2:45
6	3:22	4.1:1	6:1	2:18
7	3:02	4.5:1	7:1	1:58
8	3:19	4.1:1	8:1	1:43

Table 13 shows the change in run time as the number of processors increased. It should be noted that during the tests, there was variation in the amount of activity on the server. Using seven processors produced the best run time. Executing using all eight processors is an example of performance degradation realized by utilizing all available processors. We see from Table 13 that, practically, linear scalability is hard to achieve, but that run times can still decrease at significant rates. Compared to running on a single processor, using seven processors decreased the run time by 85%.

SUMMARY

In this paper, we discussed MIS reporting and its application in the credit card industry. Example campaign tables were created and used for showing how to manually calculate MIS reports. Then we discussed the SAS programming concepts and components used to execute the MIS report in a single I/O step in a multi-threaded environment. A %MACSUB macro was developed to exploit the SAS/ACCESS Pass-Through Facility to Oracle and MP CONNECT. It was concluded that by applying parallelism using MP CONNECT, a significant reduction in run time was achieved.

RERFERENCES

Bentley, John E. (2001), “SAS Multi-Process Connect: What, When, Where, How, and Why”, SUGI 26 Paper 269-26.

“SAS/CONNECT User’s Guide” in SAS Online Doc Version 8. Cary, NC: SAS Institute.

TRADEMARK CITATIONS

CONTACT INFORMATION

Tom Hotard
Acxiom Corporation
501-342-7329
thotar@acxiom.com

Appendix A

Code used to Generate IF-THEN and IF-THEN-ELSE Statement based on Metadata

```
data generate;
set s_soft_per_stream_all_order end=lastobs;
by st_name_order;
length str_net_if str_gro_if str_net_elements str_gro_elements $32000
       el $4;
retain str_net_if str_gro_if
       str_net_elements str_gro_elements;
retain cntstate cnt_stream 0;
if _n_=1 then do; str_net_elements=''; str_gro_elements=''; nelements=0; end;
if first.st_name_order then do;
    str_net_if = ''; str_gro_if=''; cnte = 0; cntstate=0; cnt_stream+1;
end;
if value = '0' then do;
   cntstate+1; if cntstate > 1 then el='else'; else el = '';
  str_net_if = trim(left(str_net_if))||' '||compress(el)||' if EDIT_'||compress(ed_name)||'=
1
               then n'||compress(put(all_order,z5.))||'+1;';
  str_gro_if = trim(left(str_gro_if))||' if EDIT_'||compress(ed_name)||'=1 then
               g'||compress(put(all_order,z5.))||'+1;';
      nelements+1;
 str_net_elements = trim(left(str_net_elements))||' n'||compress(put(all_order,z5.));
 str_gro_elements = trim(left(str_gro_elements))||' g'||compress(put(all_order,z5.));
end;
if last.st_name_order then do;
 call symput('netedit'||compress(cnt_stream),trim(left(str_net_if)));
 call symput('groedit'||compress(cnt_stream),trim(left(str_gro_if)));
 num_streams+1;
end;
if lastobs then do;
   call symput('net_elements',trim(left(str_net_elements)));
   call symput('gro_elements',trim(left(str_gro_elements)));
   call symput('nelements',trim(left(nelements)));
   call symput('num_streams',trim(left(num_streams)));
end;
run;
```

Appendix B

%MACSUB macro showing %SYSLPUT, SAS/ACCESS Pass-Through Facility, and MIS count code within an MP CONNECT Application

```
1  %macro macsub(mmod);
2
3  %global rmmod rschema
4          rgro_elements rnet_elements
5          rnum_streams
6          ra_ic_name ra_st_name
7          rlist;
8
9  %macro mglobal;
10  %do i = 1 %to &num_streams; %global rgroedit&i rnetedit&i; %end ;
11 %mend mglobal;
12
13 %mglobal;
14
15 signon task&mmod cmacvar=mtask&mmod cwait=no sascmd='sas';
16
17
18 %syslput rmmod=&mmod;
19 %syslput rgro_elements=&gro_elements;
20 %syslput rnet_elements=&net_elements;
21 %syslput rnum_streams=&num_streams;
22 %syslput ra_ic_name=&a_ic_name;
23 %syslput ra_st_name=&a_st_name;
24 %syslput rflist=&flist;
25
26
27 %macro msyslput;
    %do i = 1 %to &num_streams; %syslput rgroedit&i=&&groedit&i; %syslput rnetedit&i = &&netedit&i; %end
28 ;
29 %mend msyslput;
30
31 %msyslput;
32
33 rsubmit;
34
35 options nocenter mprint symbolgen macrogen merror;
36
37 libname datastor 'directory path';
38
39 PROC SQL;
40 %include 'logon script';
41 CREATE VIEW vtable AS SELECT * FROM CONNECTION TO ORACLE
```

```
(
select /*+ use_hash (CAMPAIGN.CAMP_UNIV_TBL CAMPAIGN.TAG_TBL) */
 CAMPAIGN.CAMP_UNIV_TBL.id, &rflist
from CAMPAIGN.CAMP_UNIV_TBL,
     CAMPAIGN.TAG_TBL
where CAMPAIGN.CAMP_UNIV_TBL.id = CAMPAIGN.TAG_TBL.id (+)
AND mod(CAMPAIGN.CAMP_UNIV_TBL.id,8)=&rmmod
);
QUIT;

data dunivcnt(keep=&rgro_elements &rnet_elements)
set vtable end=lastobs;
array a_ic_name(&rnum_streams) &ra_ic_name;
array a_st_name(&rnum_streams) &ra_st_name;
retain &rgro_elements
       &rnet_elements 0;
%do i = 1 %to &rnum_streams;
  if a_ic_name(&i) = 1 and a_st_name(&i) = 0 then do;
     &&rgroedit&i;
     &&rnetedit&i;
  end;
%end;
if lastobs then do;
  output dunivcnt; call symput('numobs',compress(_n_));
end;
run;

proc transpose data = dunivcnt out=tran_dunivcnt;
run;

data datastor.d&rnsub._tran_univcnt;
set tran_dunivcnt;
run;

endrsubmit;

%mend macsub;

%macsub(mmod=0);
%macsub(mmod=1);
%macsub(mmod=2);
%macsub(mmod=3);
%macsub(mmod=4);
%macsub(mmod=5);
%macsub(mmod=6);
```

```
87  %macsub(mmod=7);
88
89  listtask;
90
91  waitfor _all_ task0 task1 task2 task3 task4 task5 task6 task7;
92
93  %put mtask0= &mtask0;
94  %put mtask1= &mtask1;
95  %put mtask2= &mtask2;
96  %put mtask3= &mtask3;
97  %put mtask4= &mtask4;
98  %put mtask5= &mtask5;
99  %put mtask6= &mtask6;
100 %put mtask7= &mtask7;
101
102 rget task0; signoff task0;
103 rget task1; signoff task1;
104 rget task2; signoff task2;
105 rget task3; signoff task3;
106 rget task4; signoff task4;
107 rget task5; signoff task5;
108 rget task6; signoff task6;
109 rget task7; signoff task7;
```

Appendix C
Creating the &flist Macro Variable

```
1  libname CAMPAIGN oracle schema = CAMPAIGN user = &user pw = &passwd;
2
3  proc contents data = CAMPAIGN.CAMP_UNIV_TBL
4     out=outcont_CAMP_UNIV_TBL(keep=memname name where=(substr(name,1,3) in ('IC_','ST_')));
5  run;
6
7  proc contents data = CAMPAIGN.TAG_TBL
8     out=outcont_TAG_TBL(keep=memname name where=(substr(name,1,5) in ('EDIT_')));
9  run;
10
11 data combine;
12 merge outcont_TAG_TBL(in=a)
13       outcont_CAMP_UNIV_TBL(in=b);
14 by name;
15 if a then in_outcont_TAG_TBL = 1; else in_outcont_TAG_TBL=0;
16 if b then in_outcont_CAMP_UNIV_TBL = 1; else in_outcont_CAMP_UNIV_TBL=0;
17 run;
18
19 proc sql;
20
21 select case when index(name,'EDIT_')
22         then 'nvl('||'CAMPAIGN.'||compress(memname)||'.'||compress(name)||',0) as '||compress(name)
23         else 'CAMPAIGN.'||compress(memname)||'.'||compress(name)
24       end
25 into: flist
26 separated by ', '
27 from combine(where=(in_outcont_TAG_TBL=1 or in_outcont_CAMP_UNIV_TBL=1));
28
29 quit;
```